Introduction to Artificial Intelligence

V22.0472-001 Fall 2009

Lecture 11: Reinforcement Learning 2

Rob Fergus – Dept of Computer Science, Courant Institute, NYU

Slides from Alan Fern, Daniel Weld, Dan Klein, John DeNero

Announcements

- Assignment 2 due next Monday at midnight
- Please send email to me about final exam

2

Last Time: Q-Learning

- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory
- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar states
  - This is a fundamental idea in machine learning, and we'll see it over and over again

3

Example: Pacman

- Let's say we discover through experience that this state is bad:

- In naïve q learning, we know nothing about this state or its q states:

- Or even this one!

4

Feature-Based Representations

- Solution: describe a state using a vector of features
  - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
  - Example features:
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts
    - 1 / (dist to dot)^2
    - Is Pacman in a tunnel? (0/1)
    - …… etc.
  - Can also describe a q-state (s, a) with features (e.g. action moves closer to food)

5

Function Approximation

- Never enough training data!
  - Must generalize what is learned from one situation to other "similar" new situations
- Idea:
  - Instead of using large table to represent V or Q, use a parameterized function
    - The number of parameters should be small compared to number of states (generally exponentially fewer parameters)
  - Learn parameters from experience
  - When we update the parameters based on observations in one state, then our V or Q estimate will also change for other similar states
    - I.e. the parameterization facilitates generalization of experience

6

Linear Function Approximation

- Define a set of features f1(s), …, fn(s)
  - The features are used as our representation of states
  - States with similar feature values will be treated similarly
  - More complex functions require more complex features

$$\hat{V}_\theta(s) = \theta_0 + \theta_1 f_1(s) + \theta_2 f_2(s) + \ldots + \theta_n f_n(s)$$

- Our goal is to learn good parameter values (i.e. feature weights) that approximate the value function well
  - How can we do this?
  - Use TD-based RL and somehow update parameters based on each experience.

7

TD-based RL for Linear Approximators

1. Start with initial parameter values
2. Take action according to an explore/exploit policy
3. Update estimated model
4. Perform TD update for each parameter

$$\theta_i \leftarrow ?$$

5. Goto 2

What is a "TD update" for a parameter?

8

Aside: Gradient Descent

- Given a function $f(\theta_1,\ldots,\theta_n)$ of n real values $\theta=(\theta_1,\ldots,\theta_n)$ suppose we want to minimize f with respect to θ
- A common approach to doing this is gradient descent
- The gradient of f at point θ, denoted by $\nabla_\theta f(\theta)$, is an n-dimensional vector that points in the direction where f increases most steeply at point θ
- Vector calculus tells us that $\nabla_\theta f(\theta)$ is just a vector of partial derivatives

$$\nabla_\theta f(\theta) = \left[\frac{\partial f(\theta)}{\partial \theta_1},\ldots,\frac{\partial f(\theta)}{\partial \theta_n}\right]$$

where $\frac{\partial f(\theta)}{\partial \theta_i} = \lim_{\varepsilon\to 0} \frac{f(\theta_1,\ldots\theta_{i-1},\theta_i+\varepsilon,\theta_{i+1},\ldots,\theta_n)-f(\theta)}{\varepsilon}$

can decrease f by moving in negative gradient direction

9

Aside: Gradient Descent for Squared Error

- Suppose that we have a sequence of states and target values for each state $\langle s_1, v(s_1)\rangle, \langle s_2, v(s_2)\rangle, \ldots$
  - E.g. produced by the TD-based RL loop
- Our goal is to minimize the sum of squared errors between our estimated function and each target value:

$$E_j = \frac{1}{2}\left(\hat{V}_\theta(s_j) - v(s_j)\right)^2$$

squared error of example j; our estimated value for j'th state; target value for j'th state

- After seeing j'th state the **gradient descent rule** tells us that we can decrease error by updating parameters by:

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial E_j}{\partial \theta_i}, \qquad \frac{\partial E_j}{\partial \theta_i} = \frac{\partial E_j}{\partial \hat{V}_\theta(s_j)} \frac{\partial \hat{V}_\theta(s_j)}{\partial \theta_i}$$

learning rate

10

Aside: continued

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial E_j}{\partial \theta_i} = \theta_i - \alpha \underbrace{\left(\hat{V}_\theta(s_j) - v(s_j)\right)}_{\frac{\partial E_j}{\partial \hat{V}_\theta(s_j)}} \frac{\partial \hat{V}_\theta(s_j)}{\partial \theta_i}$$

depends on form of approximator

- For a linear approximation function:

$$\hat{V}_\theta(s) = \theta_1 + \theta_1 f_1(s) + \theta_2 f_2(s) + \ldots + \theta_n f_n(s)$$

- Thus the update becomes:
- For linear functions this update is guaranteed to converge to best approximation for suitable learning rate schedule

$$\frac{\partial \hat{V}_\theta(s_j)}{\partial \theta_i} = f_i(s_j) \qquad \theta_i \leftarrow \theta_i + \alpha\left(v(s_j) - \hat{V}_\theta(s_j)\right) f_i(s_j)$$

11

TD-based RL for Linear Approximators

1. Start with initial parameter values
2. Take action according to an explore/exploit policy

   Transition from s to s'
3. Update estimated model
4. Perform TD update for each parameter

$$\theta_i \leftarrow \theta_i + \alpha\left(v(s) - \hat{V}_\theta(s)\right) f_i(s)$$

5. Goto 2

What should we use for "target value" v(s)?

- Use the TD prediction based on the next state s'

this is the same as previous TD method only with approximation

12

$$v(s) = R(s) + \beta \hat{V}_\theta(s')$$

TD-based RL for Linear Approximators

1. Start with initial parameter values
2. Take action according to an explore/exploit policy
3. Update estimated model
4. Perform TD update for each parameter

$$\theta_i \leftarrow \theta_i + \alpha\left(R(s) + \beta \hat{V}_\theta(s') - \hat{V}_\theta(s)\right) f_i(s)$$

5. Goto 2

- Step 2 requires a model to select action
- For applications such as Backgammon it is easy to get a simulation-based model
- For others it is difficult to get a good model
- But we can do the same thing for model-free Q-learning

13

Q-learning with Linear Approximators

$$\hat{Q}_\theta(s,a) = \theta_0 + \theta_1 f_1(s,a) + \theta_2 f_2(s,a) + \ldots + \theta_n f_n(s,a)$$

Features are a function of states and actions.

1. Start with initial parameter values
2. Take action a according to an explore/exploit policy transitioning from s to s′
3. Perform TD update for each parameter

$$\theta_i \leftarrow \theta_i + \alpha\left(R(s) + \beta \max_{a'} \hat{Q}_\theta(s',a') - \hat{Q}_\theta(s,a)\right) f_i(s,a)$$

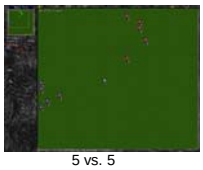
4. Goto 2

• For both Q and V, these algorithms converge to the closest linear approximation to optimal Q or V.

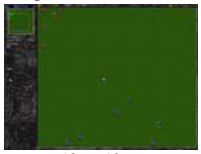
14

Example: Tactical Battles in Wargus

- Wargus is real-time strategy (RTS) game
  - Tactical battles are a key aspect of the game



5 vs. 5



10 vs. 10

- **RL Task**: learn a policy to control *n* friendly agents in a battle against *m* enemy agents
  - Policy should be applicable to tasks with different sets and numbers of agents

15

Example: Tactical Battles in Wargus

- **States**: contain information about the locations, health, and current activity of all friendly and enemy agent
- **Actions**: Attack(F,E)
  - causes friendly agent F to attack enemy E
- **Policy**: represented via Q-function Q(s,Attack(F,E))
  - Each decision cycle loop through each friendly agent F and select enemy E to attack that maximizes Q(s,Attack(F,E))
- Q(s,Attack(F,E)) generalizes over any friendly and enemy agents F and E
  - We used a linear function approximator with Q-learning

16

Example: Tactical Battles in Wargus

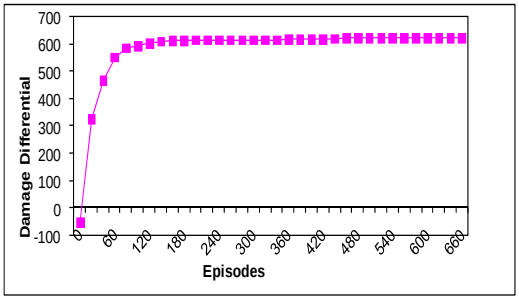
$$\hat{Q}_\theta(s,a) = \theta_1 + \theta_1 f_1(s,a) + \theta_2 f_2(s,a) + \ldots + \theta_n f_n(s,a)$$

- Engineered a set of relational features $\{f_1(s,\text{Attack}(F,E)), \ldots, f_n(s,\text{Attack}(F,E))\}$
- **Example Features**:
  - # of other friendly agents that are currently attacking E
  - Health of friendly agent F
  - Health of enemy agent E
  - Difference in health values
  - Walking distance between F and E
  - Is E the enemy agent that F is currently attacking?
  - Is F the closest friendly agent to E?
  - Is E the closest enemy agent to E?
  - …
- Features are well defined for any number of agents

17

Example: Tactical Battles in Wargus

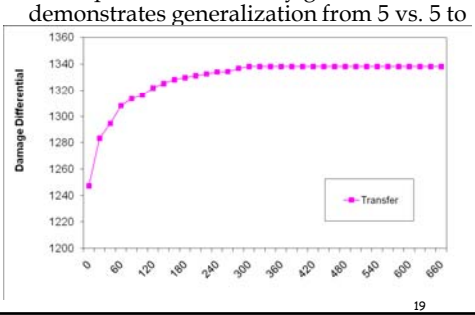
- Linear Q-learning in 5 vs. 5 battle



18

Example: Tactical Battles in Wargus

- Initialize Q-function for 10 vs. 10 to one learned for 5 vs. 5
  - Initial performance is very good which demonstrates generalization from 5 vs. 5 to



19

Q-learning w/ Non-linear Approximators

$\hat{Q}_\theta(s,a)$ is sometimes represented by a non-linear approximator such as a neural network

1. Start with initial parameter values
2. Take action according to an explore/exploit policy
3. Perform TD update for each parameter

$$\theta_i \leftarrow \theta_i + \alpha\left(R(s) + \beta \max_{a'} \hat{Q}_\theta(s',a') - \hat{Q}_\theta(s,a)\right)\frac{\partial \hat{Q}_\theta(s,a)}{\partial \theta_i}$$

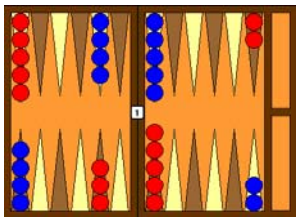
4. Goto 2

calculate closed-form

- Typically the space has many local minima and we no longer guarantee convergence
- Often works well in practice

20

~Worlds Best Backgammon Player



- Neural network with 80 hidden units
- Used TD-updates for 300,000 games against self
- Is one of the top (2 or 3) players in the world!

21

Quadruped Locomotion

- Optimize gait of 4-legged robots over rough terrain

22

RL via Policy Gradient Search

- So far all of our RL techniques have tried to learn an exact or approximate utility function or Q-function
  - I.e. learn the optimal "value" of being in a state, or taking an action from a state.
- Value functions can often be much more complex to represent than the corresponding policy
  - Do we really care about knowing Q(s,left) = 0.3554, Q(s,right) = 0.533
  - Or just that "right is better than left in state s"
- Motivates searching directly in a parameterized policy space

23

Policy Search

- Simplest policy search:
  - Start with an initial linear value function or q-function
  - Nudge each feature weight up and down and see if your policy is better than before
- Problems:
  - How do we tell the policy got better?
  - Need to run many sample episodes!
  - If there are a lot of features, this can be impractical

24

RL via Policy Gradient Descent

- This general approach has the following components
  1. Select a space of parameterized policies:
  2. Compute the gradient of the value function of the policy wrt parameters
  3. Move parameters in the direction of the gradient
  4. Repeat these steps until we reach a local maxima
- So we must answer the following questions:
  - How should we represent parameterized policies?
  - How can we compute the gradient?

25

Parameterized Policies

- One example of a space of parametric policies is:

$$\pi_\theta(s) = \arg\max_a \hat{Q}_\theta(s,a)$$

where $\hat{Q}_\theta(s,a)$ may be a linear function, e.g.

$$\hat{Q}_\theta(s,a) = \theta_0 + \theta_1 f_1(s,a) + \theta_2 f_2(s,a) + \ldots + \theta_n f_n(s,a)$$

- The goal is to learn parameters θ that give a good policy
- Note that it is not important that $\hat{Q}_\theta(s,a)$ be close to the actual Q-function
  - Rather we only require $\hat{Q}_\theta(s,a)$ is good at ranking actions in order of goodness

26

Policy Gradient Ascent

- Let $\rho(\theta)$ be the expected value of policy π_θ.
  - $\rho(\theta)$ is just the expected discounted total reward for a trajectory of π_θ.
  - For simplicity assume each trajectory starts at a single initial state.
- Our objective is to find a θ that maximizes $\rho(\theta)$
- Policy gradient ascent tells us to iteratively update parameters via:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \rho(\theta)$$

- **Problem:** $\rho(\theta)$ is generally very complex and it is rare that we can compute a closed form for the gradient of $\rho(\theta)$.
- We will instead estimate the gradient based on experience

27

Gradient Estimation

- **Concern**: Computing or estimating the gradient of discontinuous functions can be problematic.
- For our example parametric policy

$$\pi_\theta(s) = \arg\max_a \hat{Q}_\theta(s,a)$$

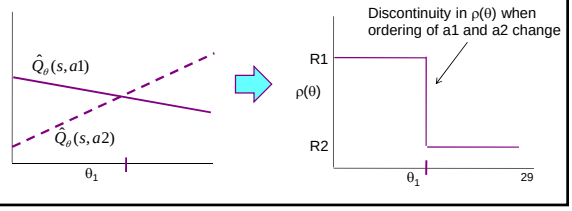
is $\rho(\theta)$ continuous?
- No.
  - There are values of θ where arbitrarily small changes, cause the policy to change.
  - Since different policies can have different values this means that changing θ can cause discontinuous jump of $\rho(\theta)$.

28

Example: Discontinous $\rho(\theta)$

$$\pi_\theta(s) = \arg\max_a \hat{Q}_\theta(s,a) = \theta_1 f_1(s,a)$$

- Consider a problem with initial state s and two actions a1 and a2
  - a1 leads to a very large terminal reward R1
  - a2 leads to a very small terminal reward R2
- Fixing θ_2 to a constant we can plot the ranking assigned to each action by Q and the corresponding value $\rho(\theta)$



29

Probabilistic Policies

- We would like to avoid policies that drastically change with small parameter changes, leading to discontinuities
- A **probabilistic policy** π_θ takes a state as input and returns a distribution over actions
  - Given a state s $\pi_\theta(s,a)$ returns the probability that π_θ selects action a in s
- Note that $\rho(\theta)$ is still well defined for probabilistic policies
  - Now uncertainty of trajectories comes from environment and policy
  - Importantly if $\pi_\theta(s,a)$ is continuous relative to changing θ then $\rho(\theta)$ is also continuous relative to changing θ
- A common form for probabilistic policies is the **softmax function or Boltzmann exploration function**

$$\pi_\theta(s,a) = \Pr(a \mid s) = \frac{\exp\left(\hat{Q}_\theta(s,a)\right)}{\sum_{a' \in A} \exp\left(\hat{Q}_\theta(s,a')\right)}$$

30

Basic Policy Gradient Algorithm

- Repeat until stopping condition
  1. Execute π_θ for N trajectories while storing the state, action, reward sequences
  2. $\nabla_\theta \leftarrow \frac{1}{N}\sum_{j=1}^{N}\sum_{t=1}^{T_j} g(s_{j,t}, a_{j,t}) R_j(s_{j,t})$
  3. $\theta \leftarrow \theta + \alpha \nabla_\theta$
- One disadvantage of this approach is the small number of updates per amount of experience
  - Also requires a notion of trajectory rather than an infinite sequence of experience
- Online policy gradient algorithms perform updates after each step in environment (often learn faster)

31

Computing the Gradient of Policy

- Both algorithms require computation of
$$g(s,a) = \nabla_\theta \log\left(\pi_\theta(s,a)\right)$$
- For the Boltzmann distribution with linear approximation we have:

where
$$\pi_\theta(s,a) = \frac{\exp\left(\hat{Q}_\theta(s,a)\right)}{\sum_{a' \in A} \exp\left(\hat{Q}_\theta(s,a')\right)}$$
$$\hat{Q}_\theta(s,a) = \theta_0 + \theta_1 f_1(s,a) + \theta_2 f_2(s,a) + \ldots + \theta_n f_n(s,a)$$

- Here the partial derivatives needed for g(s,a) are:

$$\frac{\partial \log\left(\pi_\theta(s,a)\right)}{\partial \theta_i} = f_i(s,a) - \sum_{a'} \pi_\theta(s,a') f_i(s,a')$$

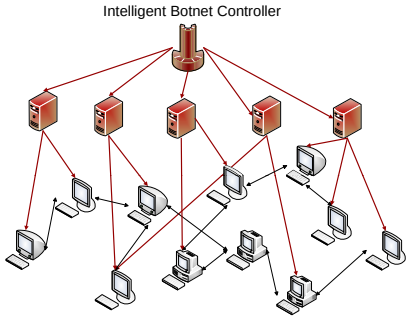
32

Controlling Helicopters

- Policy gradient techniques have been used to create controllers for difficult helicopter maneuvers
- For example, inverted helicopter flight.

33

Proactive Security



• Used OLPOMDP to proactively discover maximally damaging botnet attacks in peer-to-peer networks [Dejmal & Fern, 2008]

Policy Gradient Recap

- When policies have much simpler representations than the corresponding value functions, direct search in policy space can be a good idea
  - Allows us to design complex parametric controllers and optimize details of parameter settings
- Can be prone to finding local maxima
  - Many ways of dealing with this, e.g. random restarts.

35

Overview of AI topics

- Search
- Planning
- Logic
- Probabilities
- Machine Learning